Introduction

The Text and Voice-based Virtual Assistant project aims to develop a versatile assistant capable of interacting with users through both text and voice inputs. The assistant utilizes natural language processing (NLP) techniques to understand user commands and queries, perform various tasks, and provide responses in a user-friendly manner.

Objectives

- Develop a virtual assistant capable of understanding and responding to text and voice inputs.
- Implement features such as weather reporting, web search, task scheduling, and basic system operations.
- Utilize NLP techniques to enable natural and intuitive interaction with the virtual assistant.
- Create a user-friendly interface for seamless communication and task execution.

Tools Used

- **Programming language:** Python for backend logic and interface development.
- **Libraries and frameworks:** SpeechRecognition, tkinter for GUI, pyttsx3 for textto-speech conversion.
- **External APIs:** OpenWeather API for weather information, Google Custom Search API for web search.

Features

- Text and voice input support for user interactions.
- Weather reporting: Provides current weather information for specified locations.
- Web search: Performs online searches based on user queries.
- Task scheduling: Allows users to set reminders and schedule tasks.
- System operations: Performs basic operations like file/folder manipulation and running Python scripts.

Methodology:

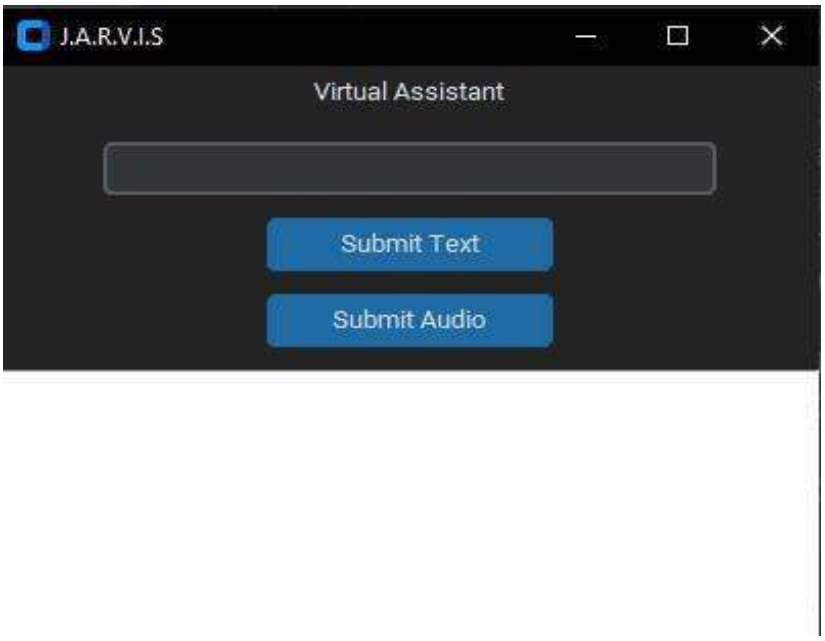
- **Requirement Analysis:** Identified key features and functionalities based on user needs and preferences.

- **Design Phase:** Designed the architecture of the virtual assistant, including text and voice input processing, task execution modules, and user interface components.
- **Implementation:** Developed the virtual assistant using Python programming language and relevant libraries/frameworks.
- **Testing:** Conducted rigorous testing to ensure the accuracy, reliability, and userfriendliness of the assistant across various scenarios.

User Interface:

The user interface is implemented using tkinter library in python, it includes two options:

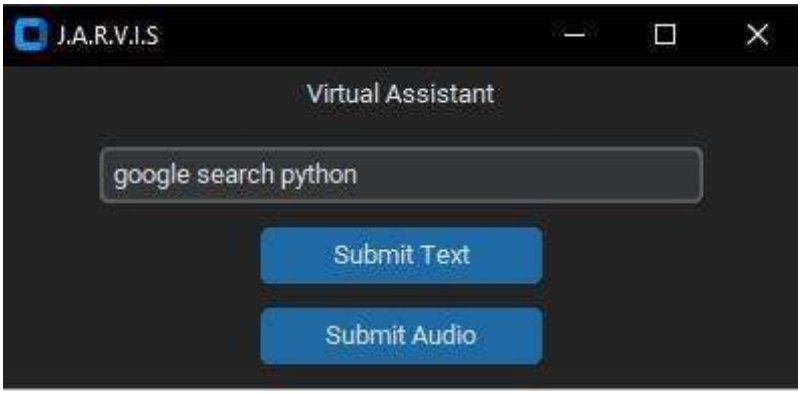
- Text search
- Audio search

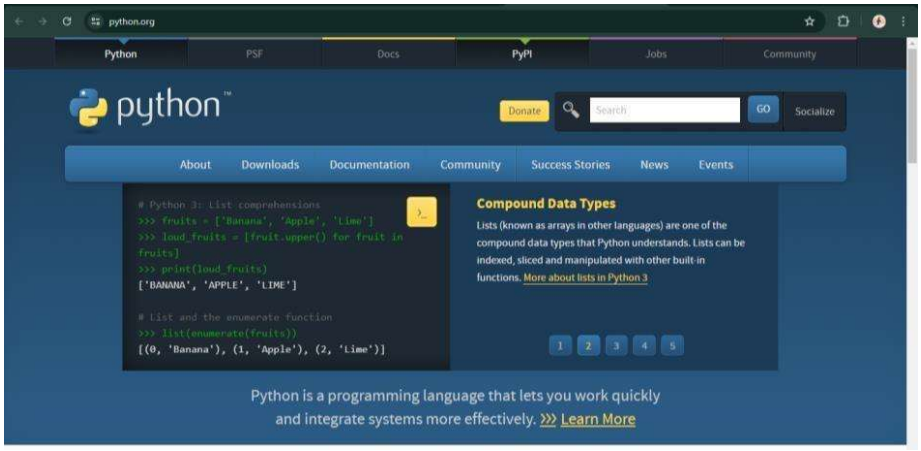


```
class VirtualAssistant:
def __init__(self, root):
        self.root = root
        self.root.title("J.A.R.V.I.S")
self.root.geometry("400x300")
set_appearance_mode('dark')
        self.label = CTkLabel(root, text="Virtual Assistant")
self.label.pack()

        self.entry = CTkEntry(root,width=300)
        # self.entry = CTkTextbox(root, height=5,width=10)
self.entry.pack(padx=10,pady=12)

        self.btn_text_input = CTkButton(root,text="Submit Text",
command=self.process_text_input)
        self.btn_text_input.pack(padx=10)

        self.btn_audio_input = CTkButton(root, text="Submit
Audio", command=self.process_audio_input)
        self.btn_audio_input.pack(padx=10,pady=12)

        self.output_text =tk.Text(root, height=10, width=50)
self.output_text.pack()

        # Initialize text-to-speech engine
self.engine = pyttsx3.init()
```

Results Analysis:

1. **Google Search:**
    - The virtual assistant provides the functionality to perform Google searches based on user queries.
    - Users can enter search queries, and the assistant conducts online searches using the Google search engine. The assistant retrieves relevant search results and presents them to the user for further exploration.
    - Utilizes the Google Custom Search API or web scraping techniques to fetch search results from Google's search engine. The assistant then displays the search results to the user, allowing them to access relevant information directly from the web.



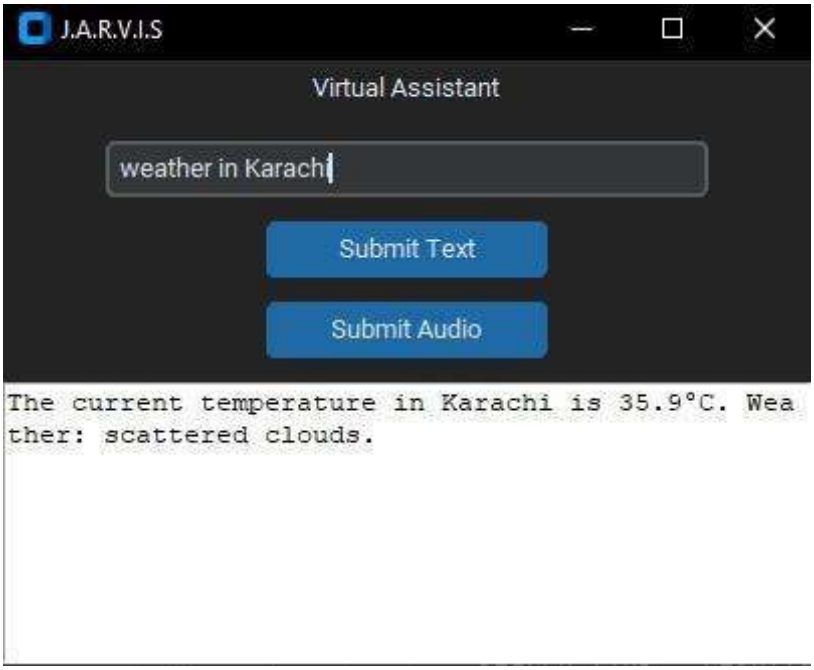


Code:

```
def googlesearch(self,query):

    url = 'https://www.googleapis.com/customsearch/v1'
params = {
        'q': query,
        'key': API_KEY,
        'cx': SEARCH_ENGINE_ID
    }
    response = requests.get(url,
params=params)      results = response.json()
if 'items' in results:
        search_link = results['items'][0]['link']
webbrowser.open(search_link)
```

2. Weather Forecast:

- The virtual assistant provides current weather information for specified locations.
- Users can inquire about the weather conditions in their desired locations, and the assistant responds with details such as temperature, humidity, wind speed, and weather forecast.
- Utilizes an external weather API, such as OpenWeather API, to fetch realtime weather data based on user queries.

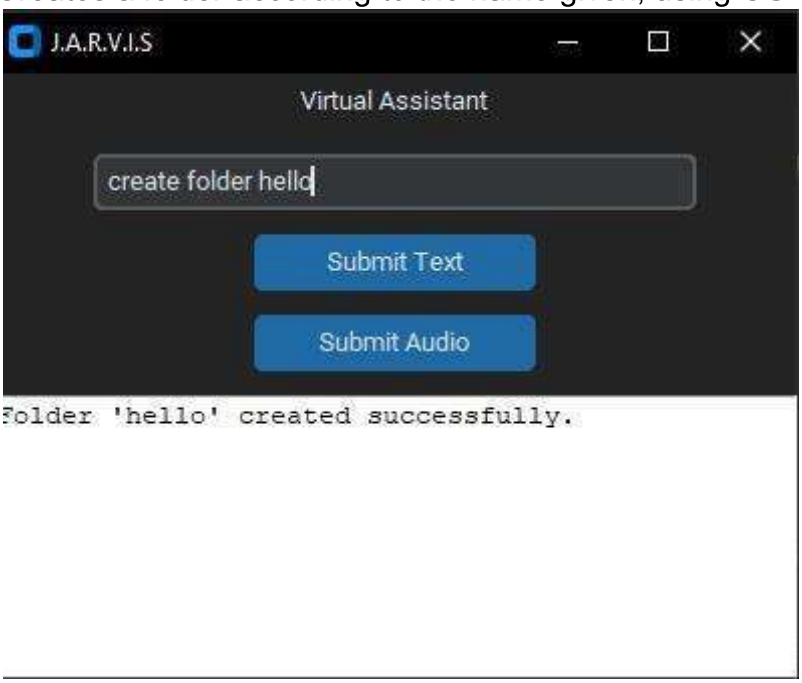


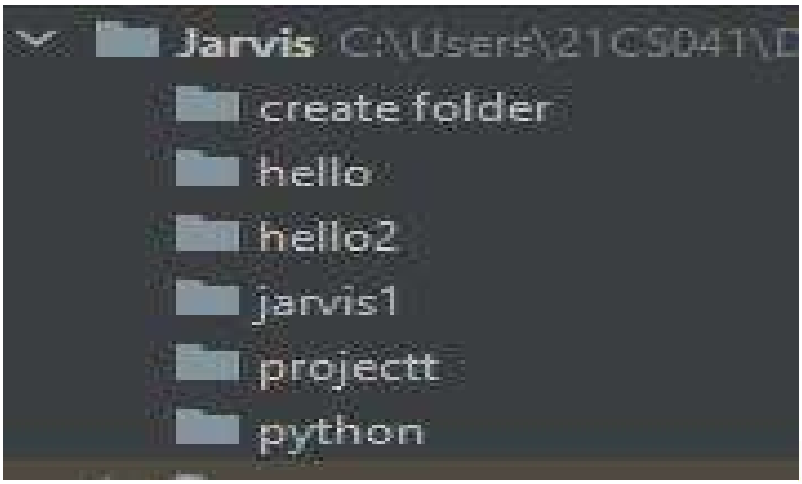
Code:

```
def get_weather(self,city):    api_key
= open('apikey2').read()
   # city = 'Hyderabad'
url =
f'http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}&units=me
tric'
   response = requests.get(url)    data = response.json()    if data['cod'] == 200:
temperature = data['main']['temp']       weather_description =
data['weather'][0]['description']       self.display_message(f"The current temperature in
{city} is {temperature}°C. Weather:
{weather_description}.")
else:
      self.display_message("Failed to fetch weather information.")
```

3. Create Folder:

Creates a folder according to the name given, using OS library functions.





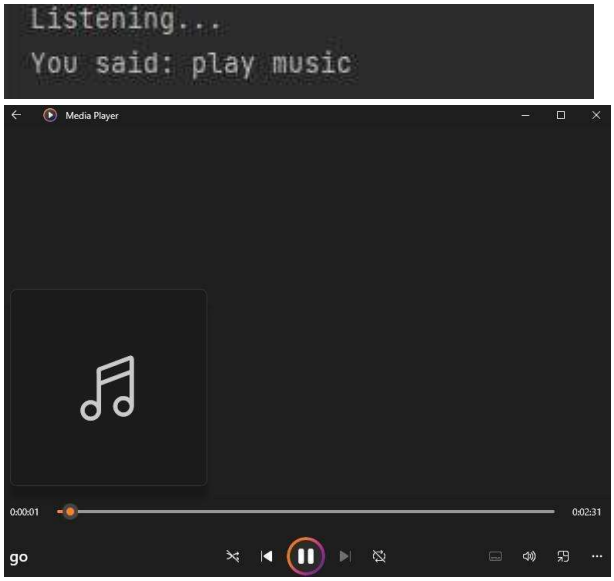
Code:

```
def create_folder(self, folder_name):
if folder_name == "create folder":
self.display_message('Try again')
else:       try:
        os.mkdir(folder_name)
        self.display_message(f"Folder '{folder_name}' created successfully.")
```

```
    except Exception as e:
        self.display_message(f"Failed to create folder '{folder_name}': {str(e)}")
```

4. Play Music:

- The virtual assistant offers the ability to play music based on user preferences.
- Integrates with music streaming services or local music libraries to access and play desired tracks.

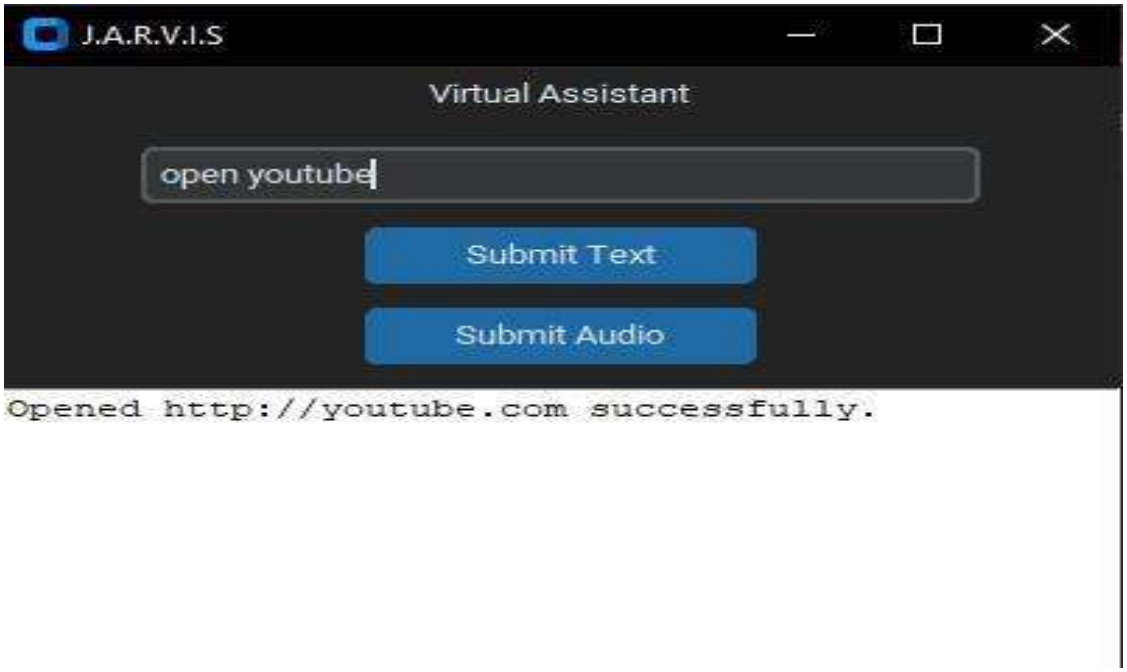


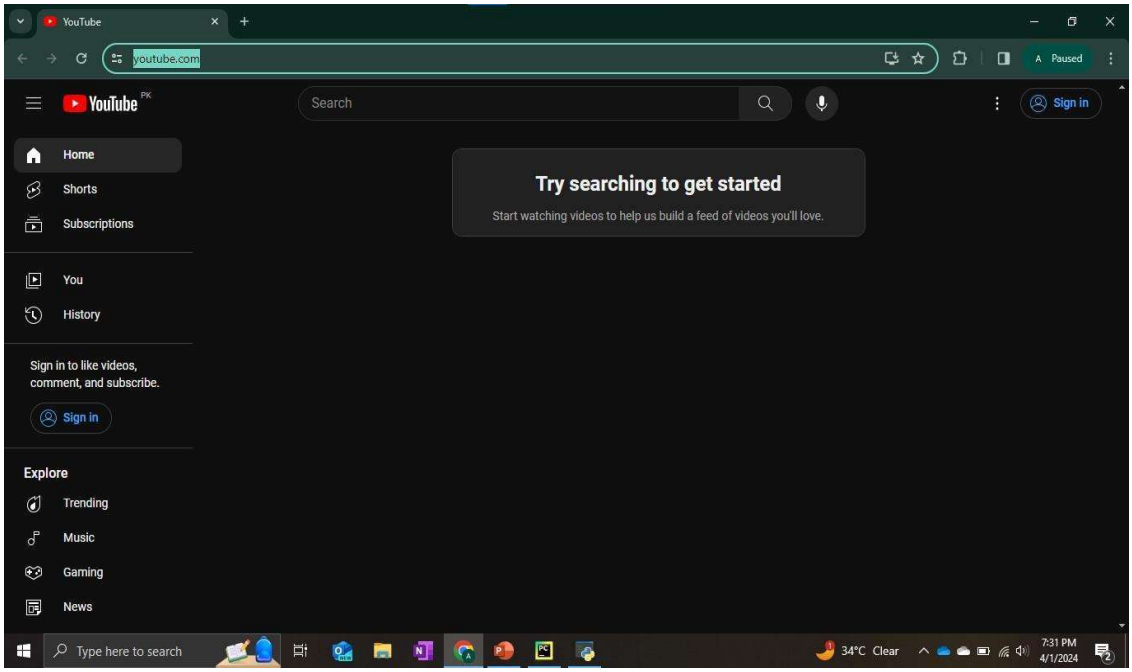
Code:

```
def play_music(self):
    musicpath = "C:/Users/21CS041/Downloads/Music/"
    music_files=os.listdir(musicpath)
    music_files = [file for file in music_files if file.endswith(('.mp3', '.wav'))]
    # Play each music file using the default system player
for music_file in music_files:
        os.system(f'start {os.path.join(musicpath, music_file)}')
```

5. Open Website:

- The virtual assistant facilitates the opening of websites based on user requests.
- Users can instruct the assistant to open specific websites by providing URLs or website names. The assistant launches the default web browser and navigates to the specified web pages.
- Utilizes web scraping techniques or predefined lists of popular websites to identify and access requested web pages.





Code:

```
def open_website(self, input_text):
    # Extract the website URL from user input
website = input_text.lower().split("open ")[-1]
    # Check if the URL starts with 'http://' or 'https://', if not, prepend it     if
not website.startswith('http://') and not website.startswith('https://'):
website = 'http://' + website     try:
```

```
    webbrowser.open(website+'.com')        self.display_message(f"Opened
{website}.com successfully.")    except Exception as e:
    self.display_message(f"Failed to open {website}: {str(e)}")
```

Future Enhancements

- Integration with more external APIs for additional functionalities.
- Advanced NLP capabilities for improved language understanding.
- Voice synthesis for natural-sounding responses.
- Multi-language support for broader user accessibility.

Conclusion:

The Text and Voice-based Virtual Assistant project demonstrates the potential of NLP and voice recognition technologies in creating intuitive and interactive user interfaces. By providing both text and voice input options, the assistant offers flexibility and convenience in accessing information and performing tasks, paving the way for more sophisticated virtual assistant applications in the future.